Technical assessment

October 23, 2018

1 Bicycle sharing demand Prediction

This is a city bicycle rented systemI'm provided Washington DC bicycle rented records per hour in two years, train datasets include every month first 19 days and test datasets consist of last 10 days(we need to predict this part of time period.

2 Data load and Analysis

we are going to use pandas in python to do data analysis ** numpy is also indispensable**

```
In [3]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
        import os
```

```
In [56]: Folder_Path = '/Users/songzhewei/Desktop/technical assessment'
         SaveFile_Name = 'DC2011-2012.csv'
         def read(path, newFileName):
             files = os.listdir(path)
             with open(path + "/" + newFileName, "w") as f:
                 for file in files:
                     if file != newFileName:
                         with open(path + "/" + file) as f1:
                             while 1:
                                 line = f1.readline()
                                 if not line:
                                     break
                                 f.write(line)
                         f.write("\n")
```

```
In [ ]: df=read(Folder_Path, SaveFile_Name)
```

```
In [58]: df1 = pd.read_csv("2011.csv")
         df2 = pd.read_csv("2012.csv")
```

```
In [59]: df=df1.append(df2)
```

load data into cacheshow it firstlet's see first 10 rows

```
In [60]: df

Out[60]:               datetime  season  holiday  workingday  weather   temp   atemp  \
         0        2011/1/1 0:00       1        0           0        1   9.84  14.395
         1        2011/1/1 1:00       1        0           0        1   9.02  13.635
         2        2011/1/1 2:00       1        0           0        1   9.02  13.635
         3        2011/1/1 3:00       1        0           0        1   9.84  14.395
         4        2011/1/1 4:00       1        0           0        1   9.84  14.395
         5        2011/1/1 5:00       1        0           0        2   9.84  12.880
         6        2011/1/1 6:00       1        0           0        1   9.02  13.635
         7        2011/1/1 7:00       1        0           0        1   8.20  12.880
         8        2011/1/1 8:00       1        0           0        1   9.84  14.395
         9        2011/1/1 9:00       1        0           0        1  13.12  17.425
         10      2011/1/1 10:00       1        0           0        1  15.58  19.695
         11      2011/1/1 11:00       1        0           0        1  14.76  16.665
         12      2011/1/1 12:00       1        0           0        1  17.22  21.210
         13      2011/1/1 13:00       1        0           0        2  18.86  22.725
         14      2011/1/1 14:00       1        0           0        2  18.86  22.725
         15      2011/1/1 15:00       1        0           0        2  18.04  21.970
         16      2011/1/1 16:00       1        0           0        2  17.22  21.210
         17      2011/1/1 17:00       1        0           0        2  18.04  21.970
         18      2011/1/1 18:00       1        0           0        3  17.22  21.210
         19      2011/1/1 19:00       1        0           0        3  17.22  21.210
         20      2011/1/1 20:00       1        0           0        2  16.40  20.455
         21      2011/1/1 21:00       1        0           0        2  16.40  20.455
         22      2011/1/1 22:00       1        0           0        2  16.40  20.455
         23      2011/1/1 23:00       1        0           0        2  18.86  22.725
         24       2011/1/2 0:00       1        0           0        2  18.86  22.725
         25       2011/1/2 1:00       1        0           0        2  18.04  21.970
         26       2011/1/2 2:00       1        0           0        2  17.22  21.210
         27       2011/1/2 3:00       1        0           0        2  18.86  22.725
         28       2011/1/2 4:00       1        0           0        2  18.86  22.725
         29       2011/1/2 6:00       1        0           0        3  17.22  21.210
         ...                ...     ...      ...         ...      ...    ...     ...
         5434  2012/12/18 18:00       4        0           1        1  15.58  19.695
         5435  2012/12/18 19:00       4        0           1        1  15.58  19.695
         5436  2012/12/18 20:00       4        0           1        1  14.76  16.665
         5437  2012/12/18 21:00       4        0           1        1  14.76  17.425
         5438  2012/12/18 22:00       4        0           1        1  13.94  16.665
         5439  2012/12/18 23:00       4        0           1        1  13.94  17.425
         5440   2012/12/19 0:00       4        0           1        1  12.30  15.910
         5441   2012/12/19 1:00       4        0           1        1  12.30  15.910
         5442   2012/12/19 2:00       4        0           1        1  11.48  15.150
         5443   2012/12/19 3:00       4        0           1        1  10.66  13.635
         5444   2012/12/19 4:00       4        0           1        1   9.84  12.120
         5445   2012/12/19 5:00       4        0           1        1  10.66  14.395
```

5446	2012/12/19 6:00	4	0	1	1	9.84	12.880
5447	2012/12/19 7:00	4	0	1	1	10.66	13.635
5448	2012/12/19 8:00	4	0	1	1	9.84	12.880
5449	2012/12/19 9:00	4	0	1	1	11.48	14.395
5450	2012/12/19 10:00	4	0	1	1	13.12	16.665
5451	2012/12/19 11:00	4	0	1	1	16.40	20.455
5452	2012/12/19 12:00	4	0	1	1	16.40	20.455
5453	2012/12/19 13:00	4	0	1	1	17.22	21.210
5454	2012/12/19 14:00	4	0	1	1	17.22	21.210
5455	2012/12/19 15:00	4	0	1	1	17.22	21.210
5456	2012/12/19 16:00	4	0	1	1	17.22	21.210
5457	2012/12/19 17:00	4	0	1	1	16.40	20.455
5458	2012/12/19 18:00	4	0	1	1	15.58	19.695
5459	2012/12/19 19:00	4	0	1	1	15.58	19.695
5460	2012/12/19 20:00	4	0	1	1	14.76	17.425
5461	2012/12/19 21:00	4	0	1	1	13.94	15.910
5462	2012/12/19 22:00	4	0	1	1	13.94	17.425
5463	2012/12/19 23:00	4	0	1	1	13.12	16.665

	humidity	windspeed	casual	registered	count
0	81	0.0000	3	13	16
1	80	0.0000	8	32	40
2	80	0.0000	5	27	32
3	75	0.0000	3	10	13
4	75	0.0000	0	1	1
5	75	6.0032	0	1	1
6	80	0.0000	2	0	2
7	86	0.0000	1	2	3
8	75	0.0000	1	7	8
9	76	0.0000	8	6	14
10	76	16.9979	12	24	36
11	81	19.0012	26	30	56
12	77	19.0012	29	55	84
13	72	19.9995	47	47	94
14	72	19.0012	35	71	106
15	77	19.9995	40	70	110
16	82	19.9995	41	52	93
17	82	19.0012	15	52	67
18	88	16.9979	9	26	35
19	88	16.9979	6	31	37
20	87	16.9979	11	25	36
21	87	12.9980	3	31	34
22	94	15.0013	11	17	28
23	88	19.9995	15	24	39
24	88	19.9995	4	13	17
25	94	16.9979	1	16	17
26	100	19.0012	1	8	9
27	94	12.9980	2	4	6

```
28          94   12.9980     2       1    3
29          77   19.9995     0       2    2
...        ...       ...   ...     ...  ...
5434        46   22.0028    13     512  525
5435        46   26.0027    19     334  353
5436        50   16.9979     4     264  268
5437        50   15.0013     9     159  168
5438        49    0.0000     5     127  132
5439        49    6.0032     1      80   81
5440        61    0.0000     6      35   41
5441        65    6.0032     1      14   15
5442        65    6.0032     1       2    3
5443        75    8.9981     0       5    5
5444        75    8.9981     1       6    7
5445        75    6.0032     2      29   31
5446        75    6.0032     3     109  112
5447        75    8.9981     3     360  363
5448        87    7.0015    13     665  678
5449        75    7.0015     8     309  317
5450        70    7.0015    17     147  164
5451        54   15.0013    31     169  200
5452        54   19.0012    33     203  236
5453        50   12.9980    30     183  213
5454        50   12.9980    33     185  218
5455        50   19.0012    28     209  237
5456        50   23.9994    37     297  334
5457        50   26.0027    26     536  562
5458        50   23.9994    23     546  569
5459        50   26.0027     7     329  336
5460        57   15.0013    10     231  241
5461        61   15.0013     4     164  168
5462        61    6.0032    12     117  129
5463        66    8.9981     4      84   88

[10886 rows x 12 columns]
```

Then we let pandas to tell us some information we have to know features name and type at the beginning

In [61]:
```
df.dtypes
```

Out[61]:
```
datetime       object
season          int64
holiday         int64
workingday      int64
weather         int64
temp          float64
atemp         float64
```

```
humidity        int64
windspeed     float64
casual          int64
registered      int64
count           int64
dtype: object
```

then we should know how large the dataset is

```
In [62]: df.shape
```

```
Out[62]: (10886, 12)
```

In conclusionwe have 10886 rowseach row has 12 different features Also there might be some noise data to deal withso let's see if there are some missing values

```
In [63]: df.count()
```

```
Out[63]: datetime      10886
         season        10886
         holiday       10886
         workingday    10886
         weather       10886
         temp          10886
         atemp         10886
         humidity      10886
         windspeed     10886
         casual        10886
         registered    10886
         count         10886
         dtype: int64
```

we can see that there is no missing values

```
In [65]: type(df.datetime)
```

```
Out[65]: pandas.core.series.Series
```

Let's process time feature, since it has much more information and target value always change with time

```
In [66]: df['month'] = pd.DatetimeIndex(df.datetime).month
         df['day'] = pd.DatetimeIndex(df.datetime).dayofweek
         df['hour'] = pd.DatetimeIndex(df.datetime).hour
```

```
In [67]: df.head(10)
```

```
Out[67]:         datetime  season  holiday  workingday  weather  temp   atemp  \
         0  2011/1/1 0:00       1        0           0        1  9.84  14.395
         1  2011/1/1 1:00       1        0           0        1  9.02  13.635
```

```
2  2011/1/1 2:00       1        0            0        1   9.02  13.635
3  2011/1/1 3:00       1        0            0        1   9.84  14.395
4  2011/1/1 4:00       1        0            0        1   9.84  14.395
5  2011/1/1 5:00       1        0            0        2   9.84  12.880
6  2011/1/1 6:00       1        0            0        1   9.02  13.635
7  2011/1/1 7:00       1        0            0        1   8.20  12.880
8  2011/1/1 8:00       1        0            0        1   9.84  14.395
9  2011/1/1 9:00       1        0            0        1  13.12  17.425

   humidity  windspeed  casual  registered  count  month  day  hour
0        81     0.0000       3          13     16      1    5     0
1        80     0.0000       8          32     40      1    5     1
2        80     0.0000       5          27     32      1    5     2
3        75     0.0000       3          10     13      1    5     3
4        75     0.0000       0           1      1      1    5     4
5        75     6.0032       0           1      1      1    5     5
6        80     0.0000       2           0      2      1    5     6
7        86     0.0000       1           2      3      1    5     7
8        75     0.0000       1           7      8      1    5     8
9        76     0.0000       8           6     14      1    5     9
```

After preprocessing time series feature, we can drop original time features And in this baseline version, we don't use registered feature as well

In [68]:
```
df_origin = df
df = df.drop(['datetime','casual','registered'], axis = 1)
```

In [69]:
```
df.head(5)
```

Out[69]:
```
   season  holiday  workingday  weather  temp   atemp  humidity  windspeed  \
0       1        0           0        1  9.84  14.395        81        0.0
1       1        0           0        1  9.02  13.635        80        0.0
2       1        0           0        1  9.02  13.635        80        0.0
3       1        0           0        1  9.84  14.395        75        0.0
4       1        0           0        1  9.84  14.395        75        0.0

   count  month  day  hour
0     16      1    5     0
1     40      1    5     1
2     32      1    5     2
3     13      1    5     3
4      1      1    5     4
```

Well, that seems more clear

In [70]:
```
df.shape
```

Out[70]:
```
(10886, 12)
```

separate dataset into two: 1. df_targetgoalcount feature 2. df_datadata

```
In [73]: df_target = df['count'].values
         df_data = df.drop(['count'],axis = 1).values
         print('df_data shape is ', df_data.shape)
         print ('df_target shape is ', df_target.shape)
```

```
df_data shape is  (10886, 11)
df_target shape is  (10886,)
```

3 Machine Learning Algorithms

the process below shows that we might spend lots of time on parameter modificationdifferent parameters would lead to different results

```
In [74]: from sklearn import linear_model
         from sklearn import cross_validation
         from sklearn import svm
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.learning_curve import learning_curve
         from sklearn.grid_search import GridSearchCV
         from sklearn.metrics import explained_variance_score
```

```
/Users/songzhewei/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:41: Depreca
  "This module will be removed in 0.20.", DeprecationWarning)
/Users/songzhewei/anaconda3/lib/python3.6/site-packages/sklearn/learning_curve.py:22: Deprecati
  DeprecationWarning)
/Users/songzhewei/anaconda3/lib/python3.6/site-packages/sklearn/grid_search.py:42: Deprecation
  DeprecationWarning)
```

data size is small, we are going to try different algorithms We would use cross validationvalidation data is 20%to see model's performancewe would try Suport Vector Regression, Ridge Regression and Random Forest Regressor

```
In [84]: cv = cross_validation.ShuffleSplit(len(df_data), n_iter=3, test_size=0.2,
             random_state=0)

         print("ridge")
         for train, test in cv:
             svc = linear_model.Ridge().fit(df_data[train], df_target[train])
             print("train score: {0:.3f}, test score: {1:.3f}\n".format(
                 svc.score(df_data[train], df_target[train]), svc.score(df_data[test], df_targe

         print ("SVR(kernel='rbf',C=10,gamma=.001)")
         for train, test in cv:

             svc = svm.SVR(kernel ='rbf', C = 10, gamma = .001).fit(df_data[train], df_target[
             print("train score: {0:.3f}, test score: {1:.3f}\n".format(
```

```
            svc.score(df_data[train], df_target[train]), svc.score(df_data[test], df_targe

    print ("Random Forest(n_estimators = 100)")
    for train, test in cv:
        svc = RandomForestRegressor(n_estimators = 100).fit(df_data[train], df_target[trai
        print("train score: {0:.3f}, test score: {1:.3f}\n".format(
            svc.score(df_data[train], df_target[train]), svc.score(df_data[test], df_targe
```

```
ridge
train score: 0.339, test score: 0.332

train score: 0.330, test score: 0.370

train score: 0.342, test score: 0.320

SVR(kernel='rbf',C=10,gamma=.001)
train score: 0.417, test score: 0.408

train score: 0.406, test score: 0.452

train score: 0.419, test score: 0.390

Random Forest(n_estimators = 100)
train score: 0.982, test score: 0.866

train score: 0.981, test score: 0.881

train score: 0.982, test score: 0.868
```

Random forest has best performance Next step we are going to do parameter modification There is tools call grid search which can help us find optimal parameter

```
In [88]: X = df_data
         y = df_target

         X_train, X_test, y_train, y_test = cross_validation.train_test_split(
             X, y, test_size=0.2, random_state=0)

         tuned_parameters = [{'n_estimators':[10,100,500]}]

         scores = ['r2']

         for score in scores:

             print(score)
```

```
            clf = GridSearchCV(RandomForestRegressor(), tuned_parameters, cv=5, scoring=score)
            clf.fit(X_train, y_train)

            print("we found optimal parameter")
            print( "")
            #best_estimator_ returns the best estimator chosen by the search
            print(clf.best_estimator_)
            print ("")
            print("score is:")
            print ("")

            for params, mean_score, scores in clf.grid_scores_:
                print("%0.3f (+/-%0.03f) for %r"
                      % (mean_score, scores.std() / 2, params))
            print( "")
```

```
r2
we found optimal parameter

RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)

score is:

0.847 (+/-0.008) for {'n_estimators': 10}
0.862 (+/-0.006) for {'n_estimators': 100}
0.863 (+/-0.006) for {'n_estimators': 500}
```

We can seeGrid Search is helpfulwe use these parameter on our model we also need to check whether our model is overfitting plot learning curve

```
In [89]: def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                                 n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):

             plt.figure()
             plt.title(title)
             if ylim is not None:
                 plt.ylim(*ylim)
             plt.xlabel("Training examples")
             plt.ylabel("Score")
             train_sizes, train_scores, test_scores = learning_curve(
                 estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
```

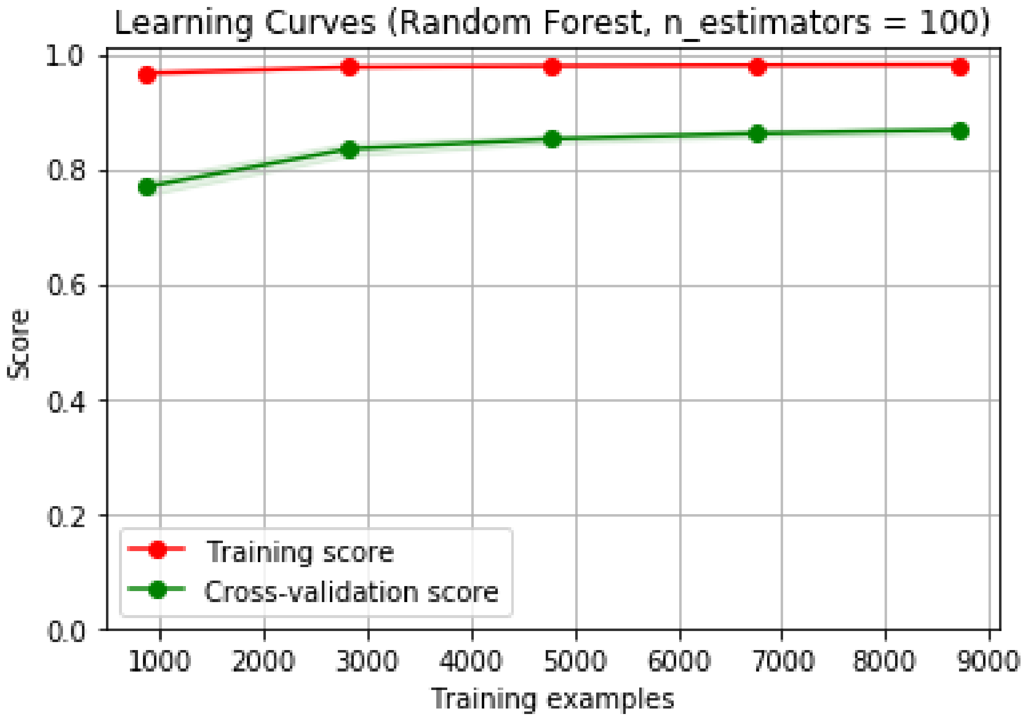
```
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt


title = "Learning Curves (Random Forest, n_estimators = 100)"
cv = cross_validation.ShuffleSplit(df_data.shape[0], n_iter=10,test_size=0.2, random_s
estimator = RandomForestRegressor(n_estimators = 100)
plot_learning_curve(estimator, title, X, y, (0.0, 1.01), cv=cv, n_jobs=4)

plt.show()
```



There is a big gap between training curve and test curve, overfiting occured

```
In [25]: # migitate overfitting
         print "Random Forest(n_estimators=200, max_features=0.6, max_depth=15)"
         for train, test in cv:
             svc = RandomForestRegressor(n_estimators = 200, max_features=0.6, max_depth=15).fi
             print("train score: {0:.3f}, test score: {1:.3f}\n".format(
                 svc.score(df_data[train], df_target[train]), svc.score(df_data[test], df_targe
```

```
/Random Forest(n_estimators=200, max_features=0.3)
train score: 0.965, test score: 0.867

train score: 0.966, test score: 0.885

train score: 0.966, test score: 0.875

train score: 0.965, test score: 0.876

train score: 0.967, test score: 0.870

train score: 0.965, test score: 0.872

train score: 0.967, test score: 0.862

train score: 0.966, test score: 0.875

train score: 0.966, test score: 0.871

train score: 0.966, test score: 0.868
```

we can use registered feature to do prediction separate dataset into two

```
In [26]: df_registered = df_origin.drop(['datetime','casual','count'], axis = 1)
         df_casual = df_origin.drop(['datetime','count','registered'], axis = 1)
```

```
In [27]: df_train_registered.head()
```

Out[27]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	\
0	1	0	0	1	9.84	14.395	81	0.0	
1	1	0	0	1	9.02	13.635	80	0.0	
2	1	0	0	1	9.02	13.635	80	0.0	
3	1	0	0	1	9.84	14.395	75	0.0	
4	1	0	0	1	9.84	14.395	75	0.0	

registered month day hour

```
0          13       1    5     0
1          32       1    5     1
2          27       1    5     2
3          10       1    5     3
4           1       1    5     4
```

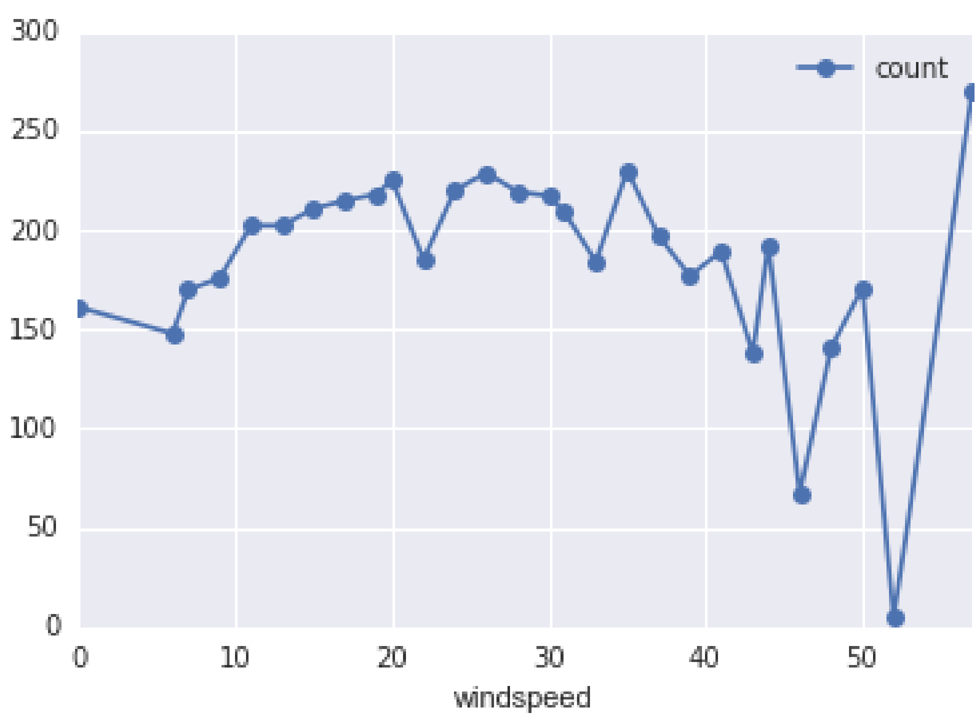
In [29]:
```
df_train_casual.head()
```

Out[29]:
```
   season  holiday  workingday  weather  temp   atemp  humidity  windspeed  \
0       1        0           0        1  9.84  14.395        81        0.0
1       1        0           0        1  9.02  13.635        80        0.0
2       1        0           0        1  9.02  13.635        80        0.0
3       1        0           0        1  9.84  14.395        75        0.0
4       1        0           0        1  9.84  14.395        75        0.0

   casual  month  day  hour
0       3      1    5     0
1       8      1    5     1
2       5      1    5     2
3       3      1    5     3
4       0      1    5     4
```

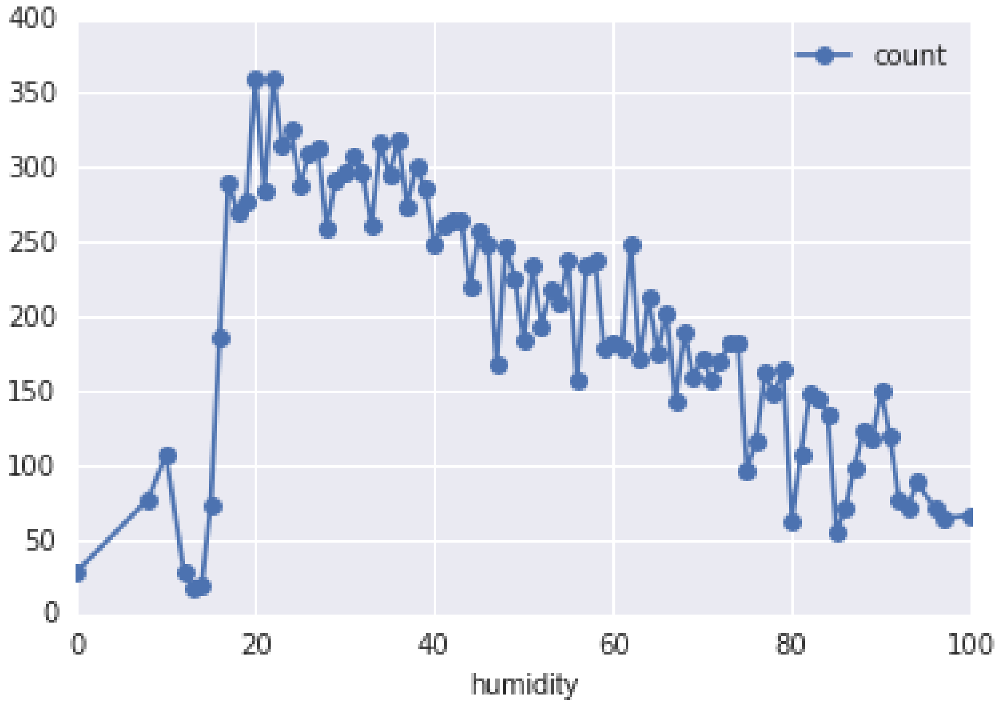
Data analysis and visulization

In [40]:
```
# windspeed
df_origin.groupby('windspeed').mean().plot(y='count', marker='o')
plt.show()
```



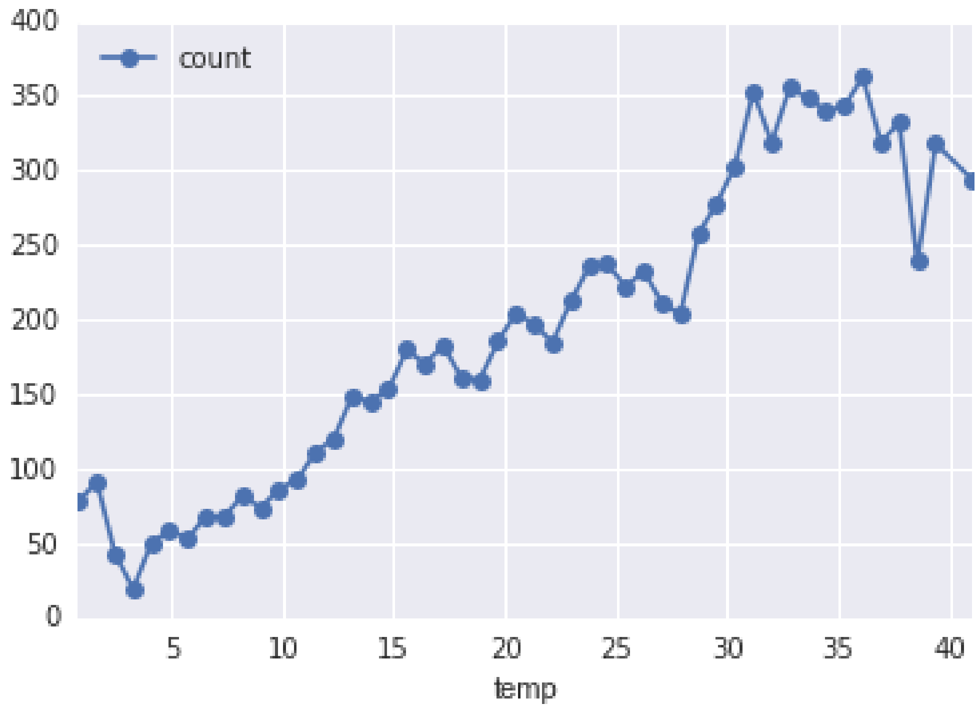
In [41]: # humidity

```
df_origin.groupby('humidity').mean().plot(y='count', marker='o')
plt.show()
```

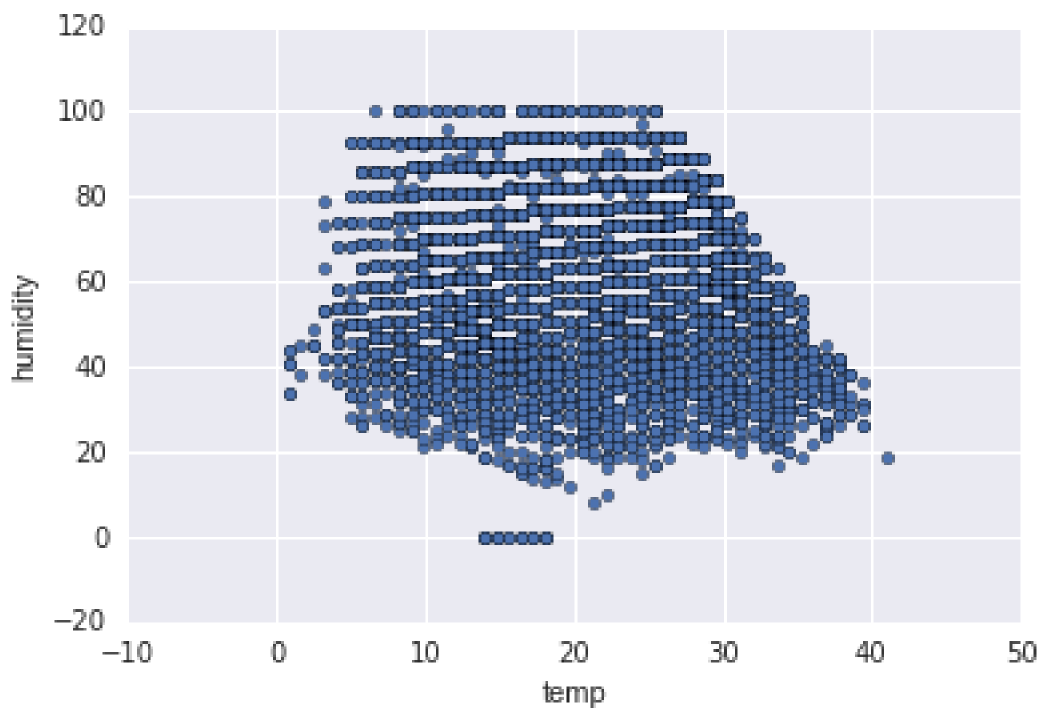


In [42]: # temperature

```
df_origin.groupby('temp').mean().plot(y='count', marker='o')
plt.show()
```

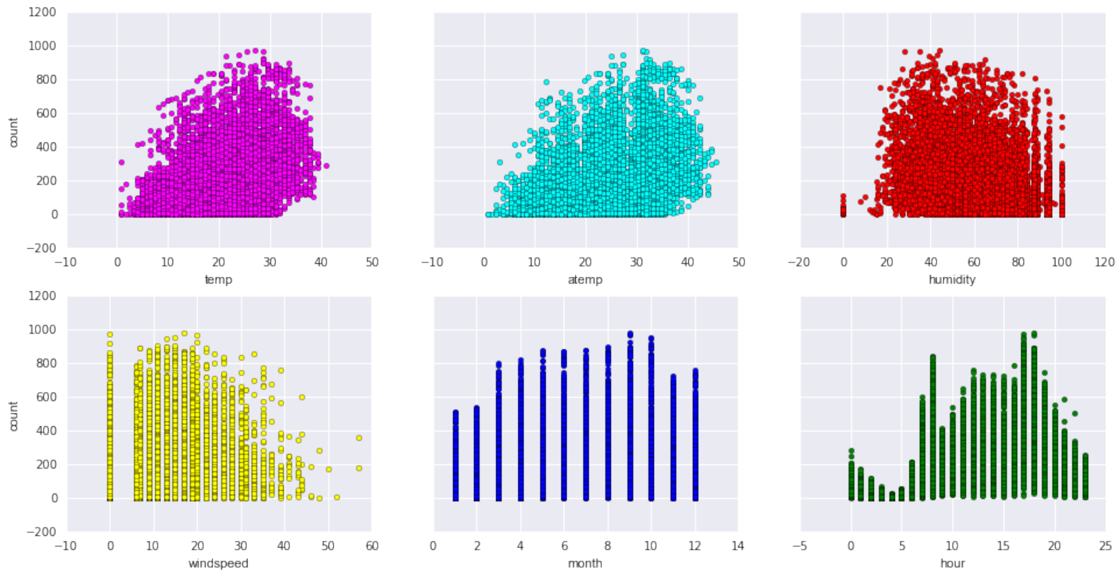



```
In [46]: #temp humidity changing
         df_train_origin.plot(x='temp', y='humidity', kind='scatter')
         plt.show()
```



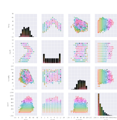
In [35]:
```
# scatter different dimentions distribution
fig, axs = plt.subplots(2, 3, sharey=True)
df_origin.plot(kind='scatter', x='temp', y='count', ax=axs[0, 0], figsize=(16, 8), col
df_origin.plot(kind='scatter', x='atemp', y='count', ax=axs[0, 1], color='cyan')
df_origin.plot(kind='scatter', x='humidity', y='count', ax=axs[0, 2], color='red')
df_origin.plot(kind='scatter', x='windspeed', y='count', ax=axs[1, 0], color='yellow')
df_origin.plot(kind='scatter', x='month', y='count', ax=axs[1, 1], color='blue')
df_origin.plot(kind='scatter', x='hour', y='count', ax=axs[1, 2], color='green')
```

Out[35]: `<matplotlib.axes._subplots.AxesSubplot at 0x11ad48090>`



In [37]:
```
sns.pairplot(df_origin[["temp", "month", "humidity", "count"]], hue="count")
```

Out[37]: `<seaborn.axisgrid.PairGrid at 0x11beecc90>`



In [48]:
```
# correlation analysis
corr = df_origin[['temp','weather','windspeed','day', 'month', 'hour','count']].corr(
corr
```

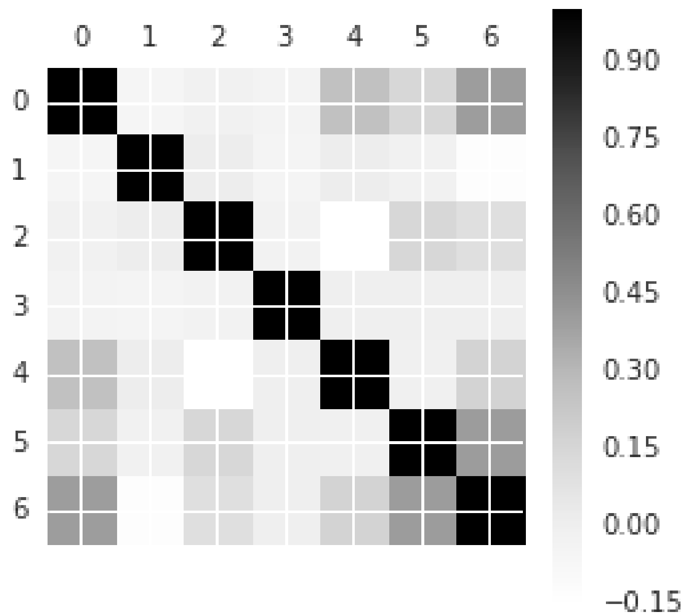
Out[48]:

	temp	weather	windspeed	day	month	hour
temp	1.000000	-0.055035	-0.017852	-0.038466	0.257589	0.145430
weather	-0.055035	1.000000	0.007261	-0.047692	0.012144	-0.022740
windspeed	-0.017852	0.007261	1.000000	-0.024804	-0.150192	0.146631
day	-0.038466	-0.047692	-0.024804	1.000000	-0.002266	-0.002925
month	0.257589	0.012144	-0.150192	-0.002266	1.000000	-0.006818
hour	0.145430	-0.022740	0.146631	-0.002925	-0.006818	1.000000
count	0.394454	-0.128655	0.101369	-0.002283	0.166862	0.400601

	count
temp	0.394454
weather	-0.128655
windspeed	0.101369
day	-0.002283
month	0.166862
hour	0.400601
count	1.000000

In [52]:
```
plt.figure()
plt.matshow(corr)
plt.colorbar()
plt.show()
```

<matplotlib.figure.Figure at 0x14716d410>


0 1 2 3 4 5 6
0 1 2 3 4 5 6
0.90
0.75
0.60
0.45
0.30
0.15
0.00
−0.15